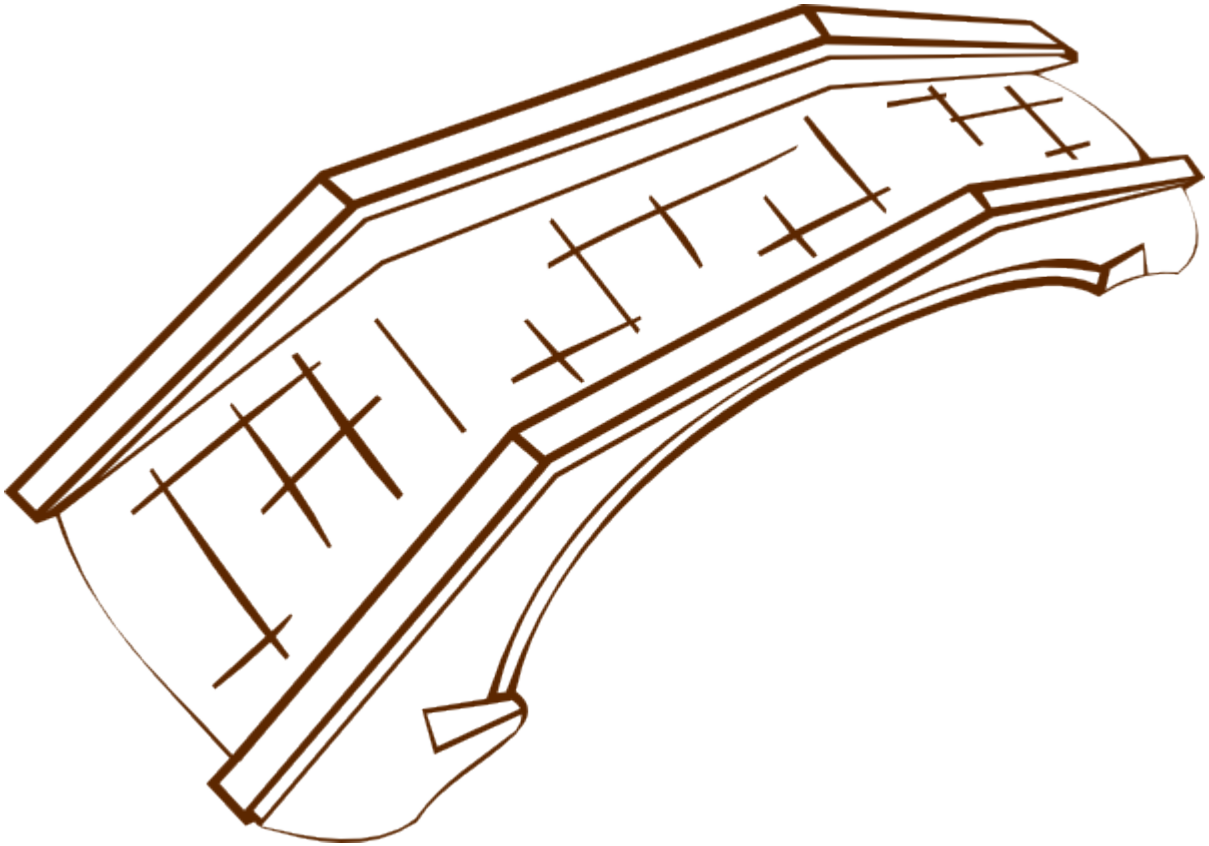
Bridge

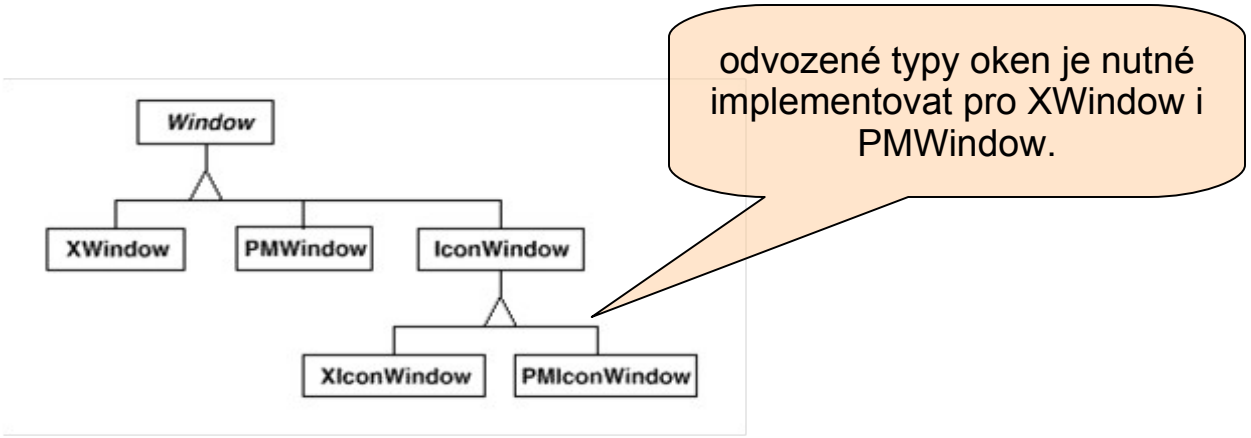


Bridge

- **Známý jako**
  - Handle/Body
- **Účel**
  - odděluje abstrakci (rozhraní a jeho sémantiku) od její konkrétní implementace
  - předchází zbytečnému nárůstu počtu tříd při přidávání implementací
  - používá se v době návrhu
- **Použitelnost**
  - odstranění vazby mezi abstrakcí a její implementací
  - abstrakci a implementaci lze nezávisle rozšiřovat
  - umožňuje měnit používanou implementaci dle potřeby
  - skrytí detailů implementace
  - při změně implementace netřeba překompilovat klientský kód

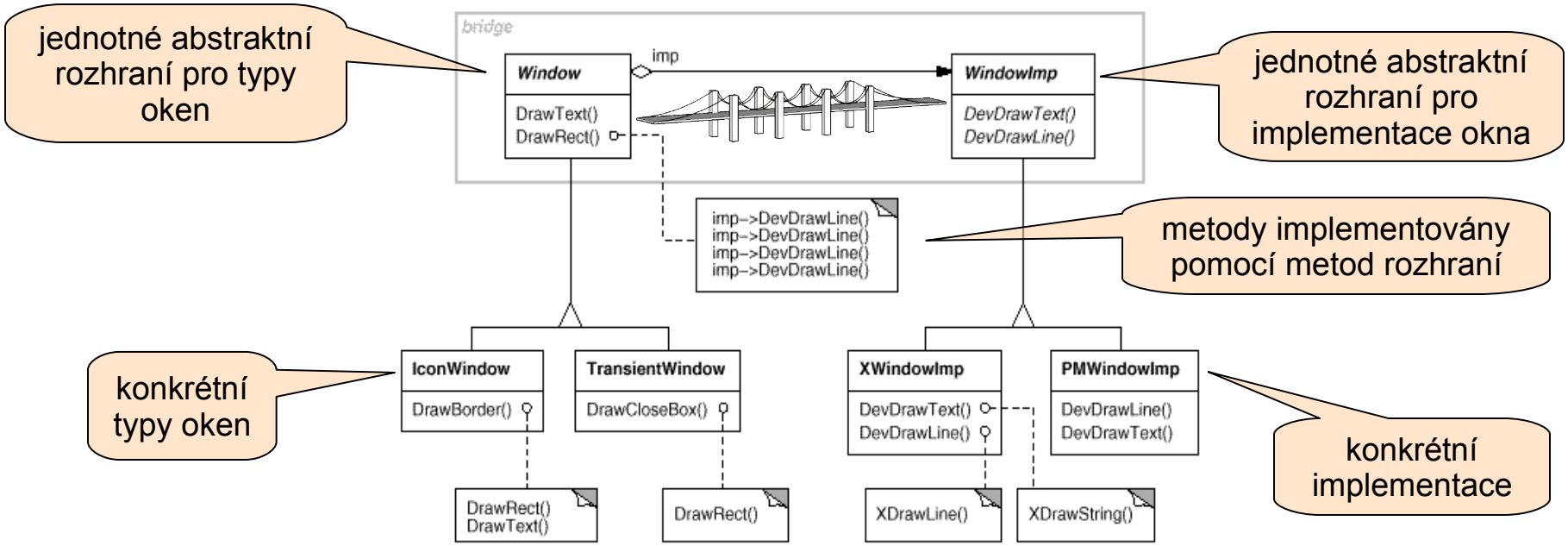
Bridge – motivace

- **Příklad: multiplatformní systém pro tvorbu grafického uživatelského rozhraní**
  - nešikovné řešení: abstraktní třída reprezentující komponentu rozhraní, pro každou platformu jeden konkrétní potomek
    - abstraktní třída `Window` reprezentující okno rozhranní, pro každou platformu jeden konkrétní potomek (`XWindow`, `PMWindow`)
    - důsledkem je zbytečný nárůst počtu tříd
    - klientský kód je závislý na platformě (jeho autor musí explicitně uvést použitou implementaci)
    - mimo to můžeme chtít různé typy oken – hlavní okno, dialogové okno, apod.



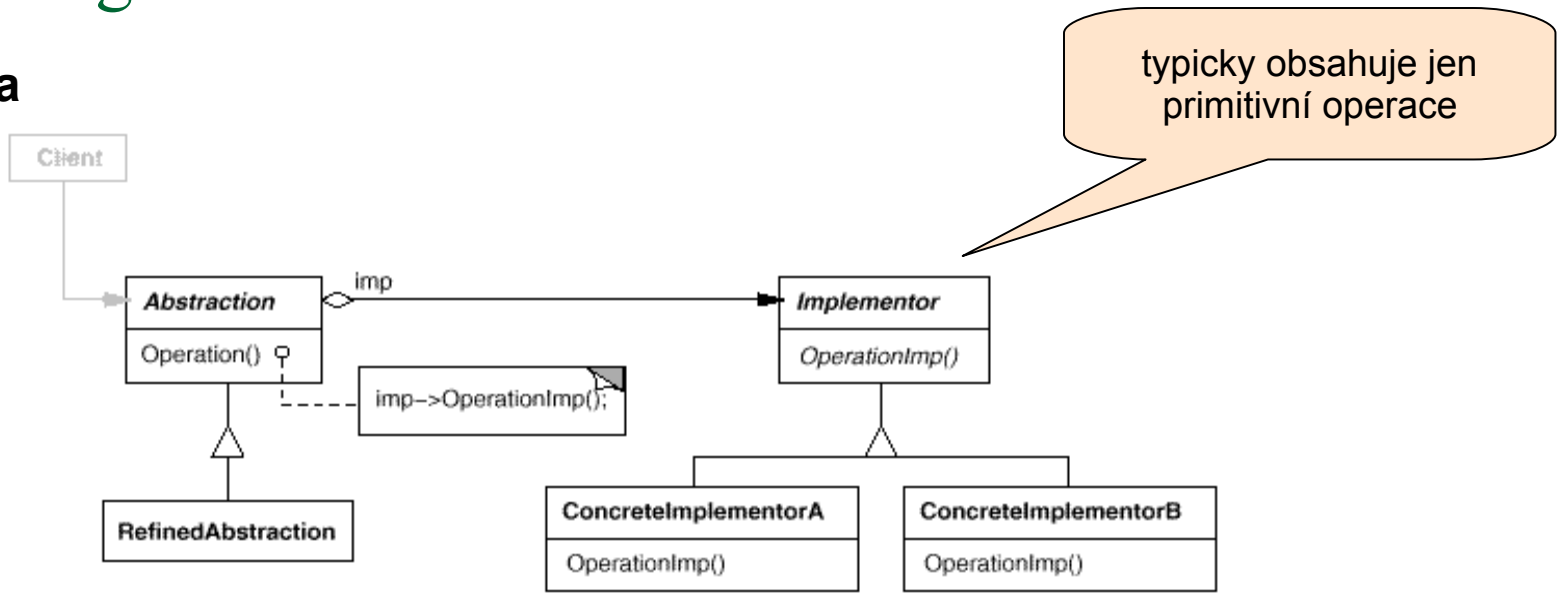
Bridge – motivace

- **Příklad: multiplatformní systém pro tvorbu grafického uživatelského rozhraní**
  - lepší řešení: vytvoříme dvě hierarchie tříd
    - 1. hierarchie: typy oken (`IconWindow`, `TransientWindow`), bázovou třídou je třída `Windows`
    - 2. hierarchie: implementace okna (`XWindowImp`, `PMWindowImp`), bázovým rozhraním (příp. abstraktní třídou) je `WindowImp`
    - třída `Window` bude obsahovat referenci na objekt typu `WindowImp`
      - metody ve třídě `Window` a jejích potomcích budou implementovány pomocí metod rozhraní `WindowImp`



Bridge – struktura

- **Struktura**



- **Účastníci**
  - Abstraction
    - definuje rozhraní objektů, obsahuje odkaz na implementaci
    - metody realizovány pomocí metod rozhraní *Implementor*
  - Implementor
    - představuje rozhraní implementací, rozhraní se může lišit od *Abstraction*
  - RefinedAbstraction
    - rozšiřuje rozhraní definované v *Abstraction*
  - ConcreteImplementor
    - konkrétní implementace rozhraní *Implementor*

Bridge – důsledky

- **Obecné vlastnosti**
  - odděluje rozhraní a implementaci
    - implementace není vázána na rozhraní, stačí implementovat jen primitivní operace
    - lepší rozšiřitelnost – obě hierarchie lze rozšiřovat nezávisle
  - silnější skrytí implementace
  - vede k lépe strukturovanému objektovému návrhu

- **Flexibilita**
  - přidání implementace
    - snadné, přidáme kód implementace a při vytváření implementace (v konstruktoru *Abstraction* nebo v *Abstract Factory*) to zohledníme
  - přidání nového potomka *Abstraction*
    - ještě jednodušší, pouze napíšeme novou třídu dědící od *Abstraction*
  - přidání potomka, který vyžaduje novou funkci
    - složitější, nutno do všech implementací přidat žádanou funkci a přidat kód potomka
    - změny v kódu jsou na dobře definovaných místech

Bridge – použití

- **Kdy je vhodné použít Bridge?**
  - nechceme, aby rozhraní a implementace byli pevně svázány
    - můžeme chtít vybrat konkrétní implementaci za běhu
  - chceme rozšiřovat rozhraní pomocí dědičnosti a zároveň přidávat nové implementace
    - Bridge umožní obě tyto věci provádět nezávisle
  - chceme, aby změny v implementaci neměly vliv na klientský kód
    - aby se tento nemusel při změně implementace rekompilovat
  - chceme skrýt detaily implementace
    - v C++ se i privátní členské proměnné a metody píší do hlavičkového souboru
  - chceme sdílet implementaci mezi více objekty
    - a nechceme, aby o tom klient věděl

Bridge – implementace

- **Varianty**
  - jeden `Implementor`
    - stačí jedna konkrétní třída `Implementor` (nepotřebujeme abstraktního předka)
    - účelem je skrytí detailů implementace (třída `Abstraction` bude obsahovat pouze referenci na objekt třídy `Implementor`, jinak nepotřebuje žádné jiné soukromé atributy)
    - pokud se změní implementace, není třeba rekompilovat klientský kód
  - více `Implementatorů`
    - Implementor se vybere v konstruktoru `Abstraction`
      - konstruktor lze parametrizovat – klient může říct, jakou implementaci použít
    - Implementor se vybere za běhu, lze ho za běhu dokonce změnit
      - implementaci může vytvářet i Abstract Factory
  - sdílení `Implementatorů`
    - Implementor je sdílen více objekty, tj. více objektů třídy `Abstraction` obsahuje referenci na tentýž objekt třídy `Implementator`
      - v C++ je potřeba použít počítání odkazů (reference counting) aby mohl být objekt třídy Implementator zrušen v případě že na něj nikdo neodkazuje

Bridge – příklad

- **Příklad: multiplatformní systém pro tvorbu grafického uživatelského rozhraní**
  - rozhraní objektů

```
public abstract class Window {
   private WindowImp imp;

   public abstract void DrawContents();

   public void DrawRect(Point p1, Point p2) {
      WindowImp imp = getWindowImp();
      imp.DeviceRect(p1.getX(), p1.getY(), p2.getX(), p2.getY());
   }

   protected WindowImp getWindowImp() {
      if (imp == null) {
         imp = WindowSystemFactory.getInstance.makeWindowImp();
      }
      return imp;
   }
}
```

implementace pomocí volání metody na konkrétní implementaci

inicializace na žádost

  - rozšířené rozhraní

```
public class IconWindow extends Window {
   public void DrawContents() {
      WindowImp imp = getWindowImp();
      imp.DeviceBitmap("icon", new Coord(0.0), new Coord(0.0));
   }
}
```

Bridge – příklad

- **Příklad: multiplatformní systém pro tvorbu grafického uživatelského rozhraní**
  - rozhraní implementací
    - poskytuje primitivní metody pro vykreslování

```
public interface WindowImp {
   void DeviceRect(Coord c1, Coord c2, Coord c3, Coord c4);
   void DeviceText(String string, Coord c1, Coord c2);
   void DeviceBitmap(String string, Coord c1, Coord c2);
   ...
}
```

  - implementace
    - třídy `XWindow` a `PMWindow` implementující rozhraní `WindowImp`, poskytují konkrétní primitivní kreslící funkce
    - mohou obsahovat privátní atributy určující stav okna

```
public class XWindowImp implements WindowImp {
   public void DeviceRect(Coord c1, Coord c2, Coord c3, Coord c4) { ... }
   ...
}

public class PMWindowImp implements WindowImp {
   public void DeviceRect(Coord c1, Coord c2, Coord c3, Coord c4) { ... }
   ...
}
```

Bridge – praktické příklady použití

- **Java Abstract Window Toolkit (AWT) – na platformě nezávislý okenní systém**
  - toolkit pro vytváření na platformě nezávislého GUI
  - třídy reprezentující komponenty grafického uživatelského rozhraní
    - abstraktní třída `Component`
      - odvozené třídy reprezentují konkrétní komponenty
    - rozhraní `ComponentPeer` definuje operace pro vykreslování komponent
      - implementující třídy - vykreslování komponent na konkrétní platformě
  - abstraktní továrna `Toolkit`
    - konkrétní továrny odvozené od této třídy vytvářejí objekty pro konkrétní platformu
  - podpora nové platformy
    - vytvořit třídy implementující rozhraní `ComponentPeer` a jeho potomky
    - implementovat továrnu odvozené od třídy `Toolkit`

Bridge – praktické příklady použití

- **Java Database Connectivity (JDBC) – připojení do databáze**
  - JDBC používá návrhový vzor Bridge k implementaci připojení do databáze
    - třída `Connection` reprezentuje připojení k databázi, které dále využívá klientský kód pro přístup k databázi
    - rozhraní `Driver` reprezentuje obecný ovladač realizující připojení k databázi
      - existují implementace tohoto rozhraní pro různé databázové systémy, tyto typicky dodávají výrobci těchto systémů
  - Abstraction Factory `DriverManager` inicializuje objekt třídy `Connection` a předá jí referenci na instanci zvoleného ovladače, tedy objekt třídy implementující rozhraní `Driver`
    - změna databáze znamená pouze výměnu ovladače
    - použitý ovladač je možné zvolit za běhu aplikace

Bridge – související návrhové vzory

- **Související návrhové vzory**
  - Abstract Factory
    - vytváří instance implementací
    - viz `java.awt.Toolkit`, `java.sql.DriverManager`
  - Adapter
    - stejně jako Bridge implementuje rozhraní jedné třídy pomocí metod jiné třídy
    - používá se v případě, že chceme nějakou již **existující** třídu přizpůsobit požadovanému rozhraní
      - tato třída má jiné rozhraní, než by se nám hodilo
    - Bridge se používá už v době návrhu tříd